Algorithmen und Datenstrukturen 2021

Prüfung zur Vorlesung als elektronische schriftliche Ausarbeitung
am 30. Juli 2021, 11:00–13:00 Uhr

Vorname	Nachname	Matrikelnummer	Prüfungscode
Otto	**Magaud**	**1**	`3GG-SS4-4J1-NIS`

- Die Prüfungsbedingungen sind online veröffentlicht. Sie haben diesen Bedinungen zugestimmt.
- Dieses Dokument enhält Ihre persönlichen Daten und darf nicht veröffentlicht werden!
- Für Aufgaben zu denen **Dateien einzureichen** sind, finden Sie **Vorgaben im Archiv** mit diesem Dokument. Diese Vorgaben sind **Teil der Aufgabenstellung**!
- Sie können insgesamt **100 Punkte (P)** erreichen.
- *Die Zeit ist knapp bemessen! Wir erwarten nicht, dass alle Aufgaben in der Zeit gelöst werden.*

Viel Erfolg!!!

Hinweise zur Bearbeitung

- Verwenden Sie **keine** Datenstrukturen aus der **Java-Standardbibliothek** mit folgenden *Ausnahmen*:
  - in der Aufgabenstellung explizit genannte Datenstrukturen,
  - in der Vorgabe enthaltene Datenstrukturen,
  - nötige Ausgaben wie z.B. `System.out.println`,
  - notwendige Interfaces (wie z.B. `Iterable` oder `Comparable`) oder Basisklassen (wie z.B. `Object`).

  Verwenden Sie insbesondere **keine Datenstrukturen/Container** aus den Java-Collections (auch nicht `ArrayList`).
- Verwenden Sie stattdessen die Datenstrukturen aus den **Code-Beispielen** zur Vorlesung!
- Sollte in einer Antwort ein **Baum** angegeben werden, verwenden Sie bitte ausschließlich die folgende, an Lisp angelehnte, **Notation**:

```
<tree>   ::= "("")"                    # leerer Baum
           | "(" <keys> <child>* ")"   # * bedeutet 0,1,2,... mal

<keys>   ::= <id> | "(" <id>* ")"      # Anzahl Schlüssel definiert Verzweigungsgrad

<child>  ::= <id>                      # Blatt mit 1 Schlüssel _ohne_ extra "(...)")
           | <tree>                    # Kinder werden von links nach rechts
                                       # angegeben, für fehlende Kinder wird
                                       # "()" verwendet

<id>     ::= Zeichenkette aus [a-zA-Z0-9]*, darf "/" als Trennzeichen
             verwenden z.B. könnte "a/r" im Rot-Schwarzbaum einen roten Knoten
             "a" bezeichnen
```

  Die Notation des Baums entsteht dann durch einen *preorder*- Durchlauf, wobei die Klammern „die Ebenen trennen“ bzw. leere Teilbäume kennzeichnen.
  Das ist die gleiche Notation, wie vorab bekannt gegeben: siehe dazu auch Hinweise und Übungsaufgabe auf den Vorlesungsseiten, wo die Notation nochmal anschaulich beschrieben ist.
  Außerdem bieten die Code-Beispiele ein Programm zur Überprüfung der Syntax.
- Halten Sie **Ihre Lösungen** möglichst *kurz und einfach*! Beschränken Sie insbesondere Änderungen von Vorgaben auf das Nötigste!

Aufgabenstellung

Aufgabe 01: Listen **5P**

Die Klasse `Lists` dient als Rahmen für eine doppelt verkettete Liste, die durch die (verschachtelte) Klasse `Node` dargestellt wird. Jeder Knoten der Liste enthält einen Wert (Attribut `data`) vom Typ **int**.

Implementieren Sie die Funktion `removeLarger(Node list,` **int** `value)`, die

- *nur einen* Durchlauf der Liste vornimmt (immer vorwärts) und dabei
- *alle* Knoten aus der Liste entfernt, deren Wert echt größer als `value` ist, und
- die neue Liste zurückgibt.

Achten Sie auf die korrekte Verkettung der Liste. Beachten Sie auch alle Sonderfälle, dabei soll **null** eine leere Liste darstellen. Wenn das Maximum aus einer leeren Liste als Eingabe entfernt werden soll, wird wieder die leere Liste zurückgegeben werden.

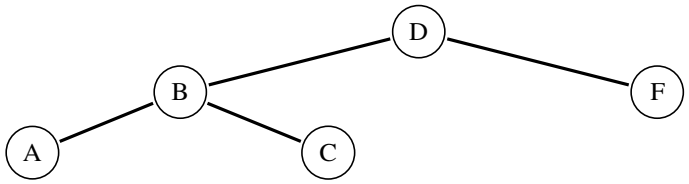
Hinweis: Die Vorgabe enthält einige Funktionen, die zum Testen nützlich sein könnten.

Reichen Sie die Datei `Lists.java` ein!

Aufgabe 02: AVL-Bäume **8P**

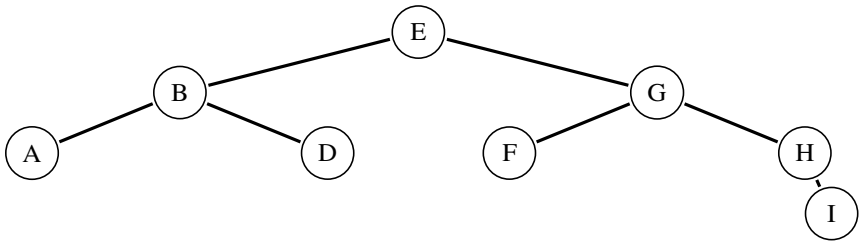
Fügen Sie in die vorgegebenen AVL-Bäume jeweils den angegebenen Wert ein. Geben Sie als Ergebnis jeweils den vollständigen AVL-Baum in der geforderten Notation (siehe *Hinweise zur Bearbeitung*) an. Der vorgegebene Baum ist jeweils auch in derselben Notation dargestellt.

`A02.Q1` Fügen Sie in den angegebenen AVL-Baum den Wert "**E**" ein.



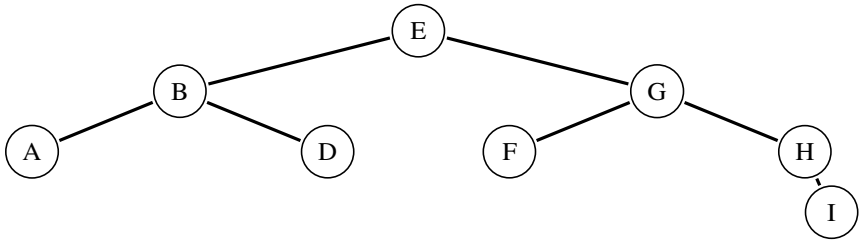

```
(D (B (A ()()) (C ()())) (F ()()))
```

`A02.Q2` Fügen Sie in den angegebenen AVL-Baum den Wert "**C**" ein.



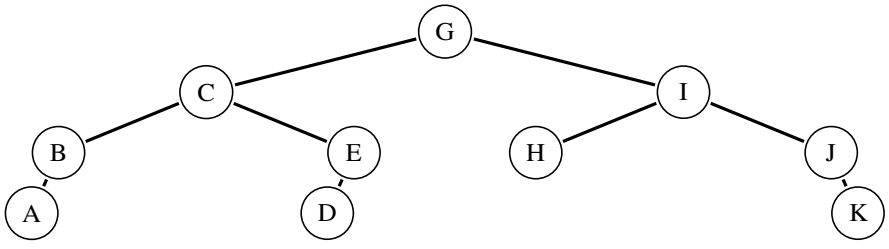

```
(E (B (A ()()) (D ()())) (G (F ()()) (H ()(I ()()))))
```

`A02.Q3` Fügen Sie in den angegebenen AVL-Baum den Wert "**C**" ein.




```
(E (B (A ()()) (D ()())) (G (F ()()) (H ()(I ()()))))
```

`A02.Q4` Fügen Sie in den angegebenen AVL-Baum den Wert "**F**" ein.

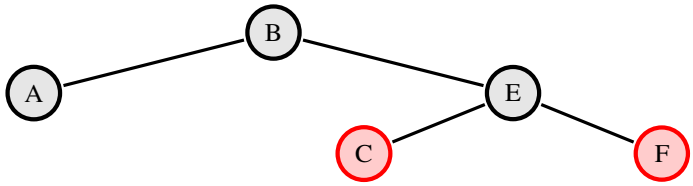



```
(G (C (B (A ()())()) (E (D ()()) ())) (I (H ()()) (J ()(K ()()))))
```

Aufgabe 03: Rot-Schwarz-Bäume **8P**

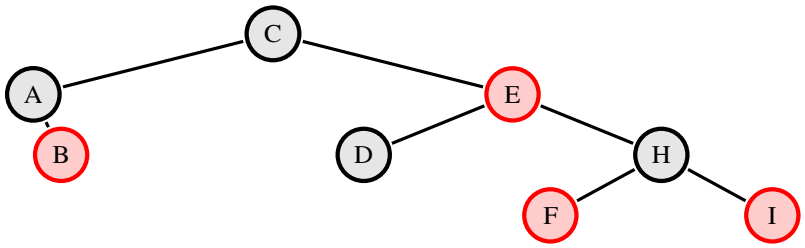
Fügen Sie in die vorgegebenen Rot-Schwarz-Bäume jeweils den angegebenen Wert ein. Geben Sie als Ergebnis jeweils den vollständigen Rot-Schwarz-Baum in der geforderten Notation (siehe *Hinweise zur Bearbeitung*) an. Kennzeichnen Sie dabei die Farbe der Knoten mit dem Suffix `/b` für schwarz (*black*) bzw. `/r` für rot. Geben Sie *keine* Hilfsknoten (Blätter ohne Schlüssel) an. *Hinweis:* Der vorgegebene Baum ist jeweils auch in derselben Notation dargestellt.

`A03.Q1` Fügen Sie in den angegebenen Rot-Schwarz-Baum den Wert "**D**" ein.



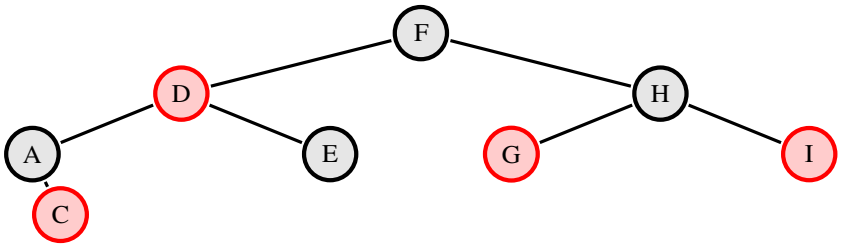

```
(B/b (A/b ()()) (E/b (C/r ()()) (F/r ()())))
```

`A03.Q2` Fügen Sie in den angegebenen Rot-Schwarz-Baum den Wert "**G**" ein.



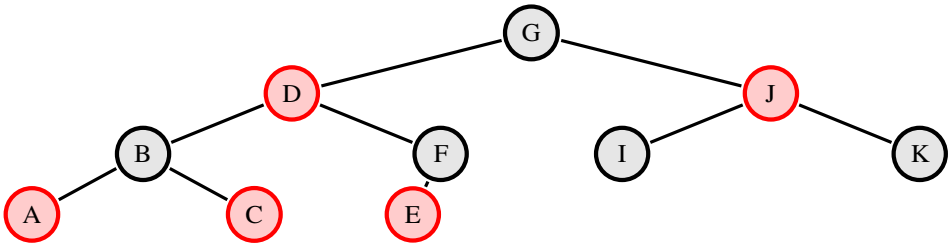

```
(C/b (A/b ()(B/r ()())) (E/r (D/b ()()) (H/b (F/r ()()) (I/r ()()))))
```

`A03.Q3` Fügen Sie in den angegebenen Rot-Schwarz-Baum den Wert "**B**" ein.




```
(F/b (D/r (A/b ()(C/r ()()))(E/b ()())) (H/b (G/r ()()) (I/r ()())))
```

`A03.Q4` Fügen Sie in den angegebenen Rot-Schwarz-Baum den Wert "**H**" ein.

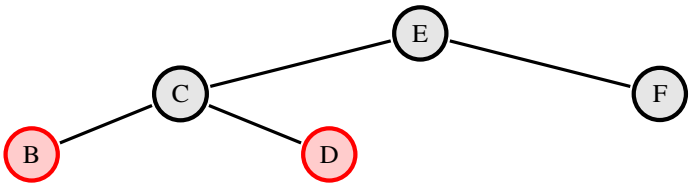



```
(G/b (D/r (B/b (A/r ()())(C/r ()()))(F/b (E/r ()()) ())) (J/r (I/b ()()) (K/b
()())))
```

Aufgabe 04: 2-3-4-Bäume **4P**

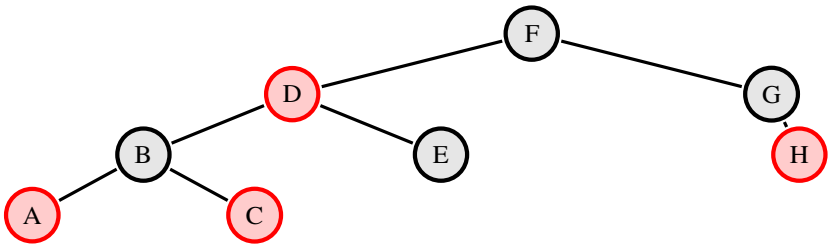
Geben Sie für jeden der Rot-Schwarz-Bäume den entprechenden 2-3-4-Baum in der geforderten Notation (siehe *Hinweise zur Bearbeitung*) an.

`A04.Q1` Geben Sie zum angegebenen Rot-Schwarz-Baum den entsprechenden 2-3-4-Baum an.




```
(E/b (C/b (B/r ()())(D/r ()())) (F/b ()()))
```

`A04.Q2` Geben Sie zum angegebenen Rot-Schwarz-Baum den entsprechenden 2-3-4-Baum an.




```
(F/b (D/r (B/b (A/r ()())(C/r ()()))(E/b ()())) (G/b ()(H/r ()())))
```

Aufgabe 05: B-Bäume **5P**

Bob implementiert einen B-Baum, dessen Knoten in einer Datei gespeichert und auch von dort gelesen werden. Dazu teilt er die Datei in *Seiten* zu 4096 Byte ein, so dass in jeder Seite ein Knoten gespeichert wird. Im B-Baum selbst sollen nur Schlüssel und Verweise auf Knoten/Seiten gespeichert werden (keine zusätzlichen Werte oder Attribute).

Bobs Anwendung sieht vor, dass jeder Schlüssel aus (höchstens) 12 Bytes besteht, und für den Verweis auf eine Seite in der Datei (deren Index) sind 6 Bytes vorgesehen.

Bob will den im Dateisystem vorhandenen Speicher möglichst effizient nutzen.

`A05.Q1` Wie viele Einträge werden mindestens in jedem Knoten gespeichert? (Geben Sie eine ganze Zahl an.)

`A05.Q2` In welchem bzw. welchen Knoten dürfen ausnahmsweise *weniger* Einträge gespeichert werden?

`A05.Q3` Wie viele Seitenzugriffe benötigt eine Suche in einem von Bobs Bäumen mit 10^{10} Einträgen im schlechtesten Fall? (Geben Sie eine ganze Zahl an.)

Aufgabe 06: Min-Heap oder nicht? 7P

Die folgenden Felder stellen teilweise binäre Min-Heaps dar.

```
int[] a = { 94, 41, 88, 28 };
int[] b = { 10, 41, 44, 29, 71 };
int[] c = { 52, 91, 99, 34, 89, 24 };
int[] d = { 18, 83, 78, 25, 93, 90, 79 };
int[] e = { 69, 99, 29, 58, 45, 39, 26, 54, 12 };
int[] f = { 11, 60, 37, 85, 94, 38, 12, 27, 72, 28, 75 };
int[] g = { 25, 72, 35, 76, 86, 87, 44, 94, 82, 51, 23, 34, 74 };
```

Geben Sie zu jedem Feld jeweils den *Index* `n` des *Kindknotens* an, an dem die *Heap-Eigenschaft* bezüglich des Elternknotens *erstmalig verletzt* ist. Falls die Heap-Eigenschaft über das gesamte Feld gilt, geben Sie stattdessen die Länge des Feldes an. *Hinweis:* Damit bezeichnet `n` die Länge eines gültigen binären Min-Heaps.

`A06.Q1` Geben Sie `n` für das Feld `a` an.

`A06.Q2` Geben Sie `n` für das Feld `b` an.

`A06.Q3` Geben Sie `n` für das Feld `c` an.

`A06.Q4` Geben Sie `n` für das Feld `d` an.

`A06.Q5` Geben Sie `n` für das Feld `e` an.

`A06.Q6` Geben Sie `n` für das Feld `f` an.

`A06.Q7` Geben Sie `n` für das Feld `g` an.

Aufgabe 07: Maximum im Min-Heap **4P**

Nehmen Sie an, ein Feld stellt einen binären Min-Heap der Länge `n` dar.

`A07.Q1` Welchen asymptotischen Aufwand (O-Notation) benötigt die Suche des *kleinsten* Eintrags?

`A07.Q2` Wie viele Einträge müssen Sie mindestens betrachten, wenn Sie den *größten* Eintrag in einem Min-Heap der Länge 779 finden wollen?

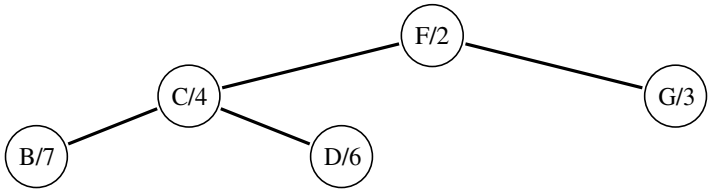
`A07.Q3` Begründen Sie kurz, wie Sie auf diese Anzahl kommen.

`A07.Q4` Welchen asymptotischen Aufwand (O-Notation) benötigt die Suche des *größten* Eintrags in Abhängigkeit von `n`?

`A07.Q5` Begründen Sie Ihre Angabe des asymptotischen Aufwands kurz.

Aufgabe 08: Suchbaum und Heap **8P**

Gegeben ist der folgende Binärbaum. Dabei stehen in jedem Knoten ein Buchstabe und eine ganze Zahl (z.B. “A/0”): der Baum ist ein *Suchbaum* mit den *Buchstaben als Schlüssel*. Gleichzeitig gilt für die Zahlen, dass die Zahl in jedem Elternknoten kleiner ist als die *Zahlen* in den Kindknoten. Das ist die *Heap-Eigenschaft* – allerdings nur die Teilordnung (ohne die Struktureigenschaft: die Ebenen müssen *nicht* vollständig gefüllt sein).




```
(F/2 (C/4 (B/7 ()())(D/6 ()())) (G/3 ()()))
```

In diesen Baum soll nun **E/1** so eingefügt werden, dass weiterhin 1. bezüglich des Buchstabens die Suchbaumeigenschaft und 2. bezüglich der Zahl die partielle Ordnung wie im binären Min-Heap gilt. Dazu fügen Sie zuerst einen neuen Blattknoten ein (wie im Suchbaum) und korrigieren danach die Position des neuen Knotens, so dass die Heap-Eigenschaft gilt.

`A08.Q1` Welche Art Operationen im Suchbaum stehen Ihnen für diese “Korrektur der Position” zur Verfügung?

`A08.Q2` Für welche Art der Traversierung des Baums ändert sich unter diesen Operationen die Reihenfolge *nicht*?

`A08.Q3` Auf welcher Ebene (beginnend von 1 an der Wurzel) muss der neu eingefügte Knoten E/1 am Ende liegen?

`A08.Q4` Welche Operationen müssen Sie nach dem Einfügen von E/1 konkret nacheinander ausführen, um die Heap-Eigenschaft wieder herzustellen? Nennen Sie diese präzise (aber ohne Bezug zu Knoten) durch Komma getrennt.

`A08.Q5` Welcher Baum ergibt sich insgesamt nach dem Einfügen von E/1, nachdem die Heap-Eigenschaft wieder hergestellt wurde?
Geben Sie als Ergebnis den vollständigen Baum in der geforderten Notation (siehe *Hinweise zur Bearbeitung*) an.

Aufgabe 09: Lineares Sondieren **5P**

Fügen Sie die Werte 83, 37, 11, 72, 33 in dieser Reihenfolge nacheinander in eine Hashtabelle der Länge 7 ein. Verwenden Sie als Hashfunktion $h(x) = x \bmod 7$. Lösen Sie Kollisionen durch *lineares Sondieren* mit $h(x, i) = h(x) + 5 \cdot i$ für die i-te Kollision.

A09.Q1 Wie sieht die Tabelle nach dem Einfügen der Werte aus?
Geben Sie die Einträge der Tabelle der Reihenfolge nach durch Komma getrennt an, für jeden nicht verwendeten Eintrag schreiben Sie dabei bitte "$*$".

A09.Q2 Bob hat in der Vorlesung nicht aufgepasst. Er verwendet eine Tabelle der Länge 205 und dieselbe Hashfunktion und dieselbe lineare Sondierung wie oben beschrieben. Bob wundert sich: Die Tabelle scheint zu schnell voll zu werden.
Erklären Sie Bob in Ihrer Antwort kurz, warum dieser Effekt eintritt und was er falsch gemacht hat.

Aufgabe 10: Hashfunktion **5P**

Die Klasse `HashWanted` beschreibt eine Person mit verschiedenen Eigenschaften/Attributen.

Nehmen Sie an, Instanzen von `HashWanted` sollen in eine Hashtabelle eingefügt werden. Geben Sie eine sinnvolle Implementierung der Methode `hashCode` an!

Hinweise:

- Sie sollen *keine* Hashtabelle implementieren!
- Für den Datentyp `String` ist `hashCode` bereits implementiert.
- Sie können Hashwerte h_1, h_2 einfach durch Addition (oder in linearen Kombinationen) oder durch exklusives Oder (XOR, in Java `h1^h2`) miteinander verknüpfen.

Reichen Sie die Datei `HashWanted.java` ein!

Aufgabe 11: Keine gute Idee?! 2P

Die folgende Methode soll eine Hashfunktion für die Klasse `String` bezeichnen. (Uns interessiert hier nur die Methode selbst.)

```
1  // method of class String
2  public int hashCode() {
3    int hash=0;
4    int skip=Math.max(1,length()/8);
5
6    for (int i=0;i<length();i+=skip)
7       hash=(hash*37)+charAt(i);
8
9    return hash;
10 }
```

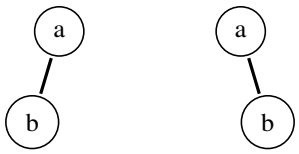
A11.Q1 Warum ist diese Hashfunktion *nicht* gut? Was sollte geändert werden?

Aufgabe 12: Hashfunktion für Binärbaum **5P**

Die Klasse `HashDataStructure` dient als Rahmen, um einen Binärbaum in der (verschachtelten) Klasse `TreeNode` zu definieren. Dabei hält jeder Knoten eine Datenwert `data` vom Typ `String`, außerdem stellt jeder Knoten rekursiv auch einen binären (Teil-)Baum dar.

Nehmen Sie an, Sie wollen Binärbäume in eine Hashtabelle einfügen. Dabei sollen zwei Bäume als *gleich* angesehen werden, genau dann wenn sie sowohl in den Datenwerten als auch in der Baumstruktur (d.h. Position der Kindknoten) exakt übereinstimmen.

Beispiel: Die folgenden Bäume sind *nicht* gleich



- Implementieren Sie die Berechnung einer sinnvollen Hashfunktion für `TreeNode` wie in Java gefordert durch die Methode `hashCode`. Der Knoten stellt dabei die Wurzel eines Baums dar. Verwenden Sie dabei *keine sekundären Datenstrukturen* z.B. eine Darstellung über einer Liste oder eine Umwandlung in `String`!
  Achten Sie auf eine möglichst effiziente Implementierung, durchlaufen Sie den Baum nicht öfter als nötig.
- Welche weitere Methode müssten Sie implementieren? *Antworten Sie in einem Kommentar! Es ist* keine Implementierung *nötig.*

Hinweise:

- Für den Datentyp `String` ist `hashCode` bereits implementiert.
- Sie können Hashwerte h_1, h_2 einfach durch Addition (oder in linearen Kombinationen) oder durch exklusives Oder (XOR, in Java `h1^h2`) miteinander verknüpfen.

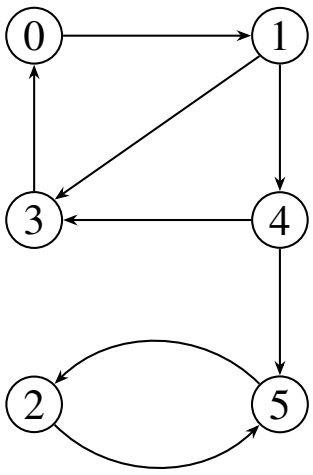
Reichen Sie die Datei `HashDataStructure.java` ein!

Aufgabe 13: Zyklen in gerichteten Graphen 10P

Diese Aufgabe befasst sich mit *gerichteten Graphen*, deren Darstellung als *Kantenliste* und *Adjazenzmatrix* sowie *Zyklen* in Graphen. Die Datei `DirectedGraphCycle.java` beinhaltet das Grundgerüst der entsprechenden Klasse `DirectedGraphCycle` sowie eine Implementierung der Klasse `Matrix`. Vervollständigen Sie die Klasse `DirectedGraphCycle` um folgende Methoden:

- **int**`[][] edgeListToMatrix(`**int**`[] edgeList)` nimmt eine Kantenliste eines *gerichteten* Graphen als **int**`[]`-Array entgegen und gibt die entsprechende Adjazenzmatrix als **int**`[][]`-Array zurück.
- **boolean** `containsCycles(`**int**`[][] adjMat,` **int** `cycleLength)` bekommt als Argument die vorher berechnete Adjazenzmatrix und eine Zyklus-Länge. Die Methode prüft, ob der entsprechende Graph einen Zyklus mit der gegebenen Länge besitzt. Falls ja wird **true** zurück gegeben.
- **int** `getCyclesCount(`**int**`[][] adjMat,` **int** `cycleLength)` berechnet die Anzahl der möglichen *Startknoten*, für die ein Zyklus der gegebenen Länge im Graph existiert.
- `Vector<Integer> getCycleStartNodes(`**int**`[][] adjMat,` **int** `cycleLength)` gibt einen `Vector` mit den Indizes von möglichen *Startknoten* für jeden Zyklus mit der gegebenen Länge `cycleLength` zurück. (*Hinweis:* Der `Vector` enthält *einen* Startknoten pro Zyklus.)

Der folgende gerichtete Graph ist über die Kantenliste `{6, 8, 0, 1, 1, 3, 1, 4, 3, 0, 4, 3, 4, 5, 5, 2, 2, 5}` definiert und dient als Beispiel. Wenn Sie alles korrekt implementiert haben, sollte die Ausgabe rechts auf der Konsole erscheinen.



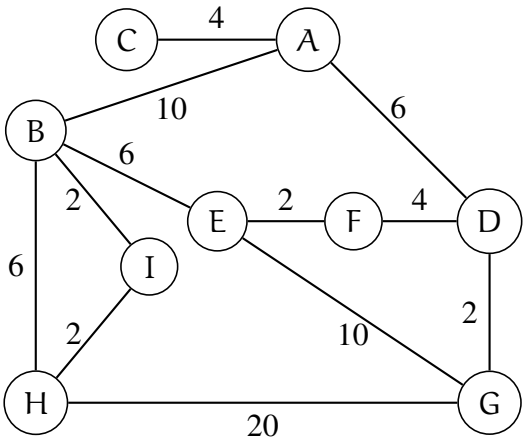

```
Startknoten für Zyklen der Länge 0: ---
Startknoten für Zyklen der Länge 1: ---
Startknoten für Zyklen der Länge 2: 2 5
Startknoten für Zyklen der Länge 3: 0 1 3
Startknoten für Zyklen der Länge 4: 0 1 2 3 4 5
Startknoten für Zyklen der Länge 5: ---
Startknoten für Zyklen der Länge 6: 0 1 2 3 5
```

Hinweis: Sie können die Klasse `Matrix` verwenden. Diese ist innerhalb von `DirectedGraphCycle.java` implementiert und bietet grundlegende Matrix-Operationen, wie *Matrix-Matrix-Multiplikation*, *transponieren*, *Zugriff auf Spalten/Zeilen*, etc. (Ändern Sie nicht die Signaturen der zu implementierenden Methoden!)

Reichen Sie die Datei `DirectedGraphCycle.java` ein!

Aufgabe 14: Kürzeste Pfade und minimaler Spannbaum 12P

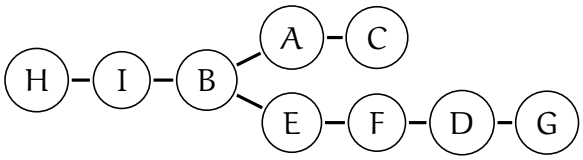
Gegeben ist der folgende gewichtete Graph.



Die nachfolgend abgebildeten Bäume sind *Spannbäume* dieses Graphen mit unterschiedlichen Eigenschaften. Kennzeichnen Sie jeden dieser Bäume je nach Eigenschaft als **SPT** (*shortest path tree* aus dem Algorithmus von Dijkstra), **MST** (minimalen Spannbaum aus dem Algorithmus von Prim), **DFS** (Spannbaum aus einer Tiefensuche ohne Gewichte), **BFS** (Spannbaum aus einer Breitensuche ohne Gewichte) oder auch "**anders**", falls keine der vorherigen Beschreibungen zutrifft.

Beachten Sie, dass die Anordnung der Kindknoten auf gleicher Ebene von links nach rechts willkürlich gewählt ist, und dass die Bäume je nach Höhe um 90° gedreht sind (mit Wurzel links).

A14.Q1 Kennzeichnen Sie den folgenden Baum!



A14.Q1.A BFS

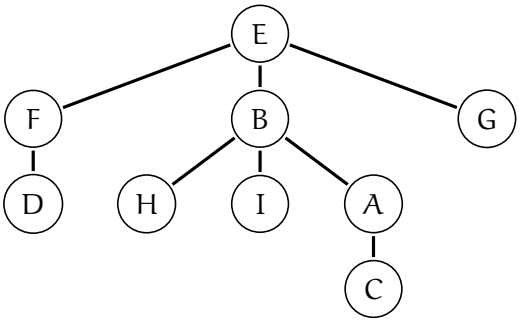
A14.Q1.B MST

A14.Q1.C SPT

A14.Q1.D DFS

A14.Q1.E anders

A14.Q2 Kennzeichnen Sie den folgenden Baum!



A14.Q2.A BFS

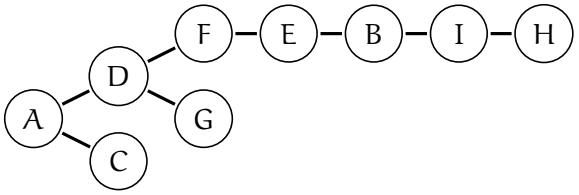
A14.Q2.B MST

A14.Q2.C SPT

A14.Q2.D DFS

A14.Q2.E anders

A14.Q3 Kennzeichnen Sie den folgenden Baum!



A14.Q3.A BFS

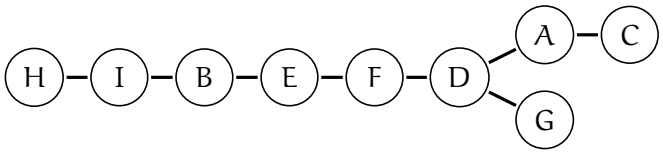
A14.Q3.B MST

A14.Q3.C SPT

A14.Q3.D DFS

A14.Q3.E anders

A14.Q4 Kennzeichnen Sie den folgenden Baum!



A14.Q4.A BFS

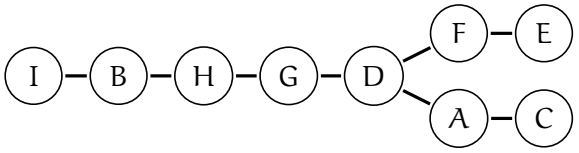
A14.Q4.B MST

A14.Q4.C SPT

A14.Q4.D DFS

A14.Q4.E anders

A14.Q5 Kennzeichnen Sie den folgenden Baum!



A14.Q5.A BFS

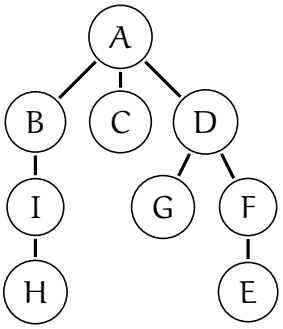
A14.Q5.B MST

A14.Q5.C SPT

A14.Q5.D DFS

A14.Q5.E anders

A14.Q6 Kennzeichnen Sie den folgenden Baum!



A14.Q6.A BFS

A14.Q6.B MST

A14.Q6.C SPT

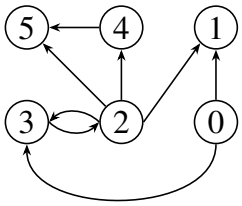
A14.Q6.D DFS

A14.Q6.E anders

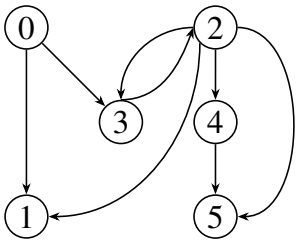
Aufgabe 15: Graphen und Matrizen **8P**

Gegeben sind folgende Graphen und Adjazenz-Matrizen:

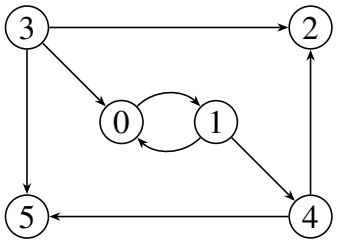
G1 :



G2 :



G3 :



M1 : $\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

M2 : $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

M3 : $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

`A15.Q1` Geben Sie an, welcher Graph durch welche Adjazenzmatrix dargestellt wird! Verwenden Sie die Notation G#M#, G#M#, ... (z.B. G4M5, G6M8 für Graph4/Matrix5, Graph6/Matrix8)!

`A15.Q2` Wieviele Quellen hat der Graph, der zu M2 gehört?

`A15.Q3` Wieviele Senken hat der Graph, der zu M2 gehört?

`A15.Q4` Geben Sie die Kantenliste für M1 als **int** `[]`-Array an!
(Schreibweise wie auf den Folien, z.B. **int** `[] edgelist={2, 2, 0, 1, 1, 0};`)

Aufgabe 16: Unendliche Weiten **4P**

Alice will ein Computerspiel mit einer potentiell unendlich großen Spielwelt implementieren. Jeder Ort im Spiel wird durch ganzzahlige Koordinaten (i, j) bezeichnet. Von jedem Ort aus können die 8 Nachbarorte $(i \pm a, j \pm b)$ und $(i \pm b, j \pm a)$ für $a \neq b$ erreicht werden. (Für $a = 1, b = 2$ entspricht dies der Bewegung des Springers beim Schach.)

Alice will Erreichbarkeit von Orten und kürzeste Wege zwischen zwei Orten mit Hilfe des Algorithmus von Dijkstra und später vielleicht sogar mit dem $A^\star$-Algorithmus berechnen. Allerdings ist ihr Graph potentiell unendlich groß, und sie weiß vorab nichts über die Start- und Zielkoordinaten (insbesondere weder Ober- noch Untergrenzen).

Alice überlegt, wie sie den Graphen darstellen könnte. Dabei will sie z.B. beim Ausführen des Dijkstra-Algorithmus möglichst effizient sowohl mit Speicher als auch mit Rechenzeit umgehen.

(Es gäbe noch weitere Probleme zu lösen, aber die heben wir uns für eine andere Prüfung auf.)

`A16.Q1` Skizzieren Sie, wie Alice den Graphen effizient darstellen könnte und begründen Sie Ihren Ansatz kurz.